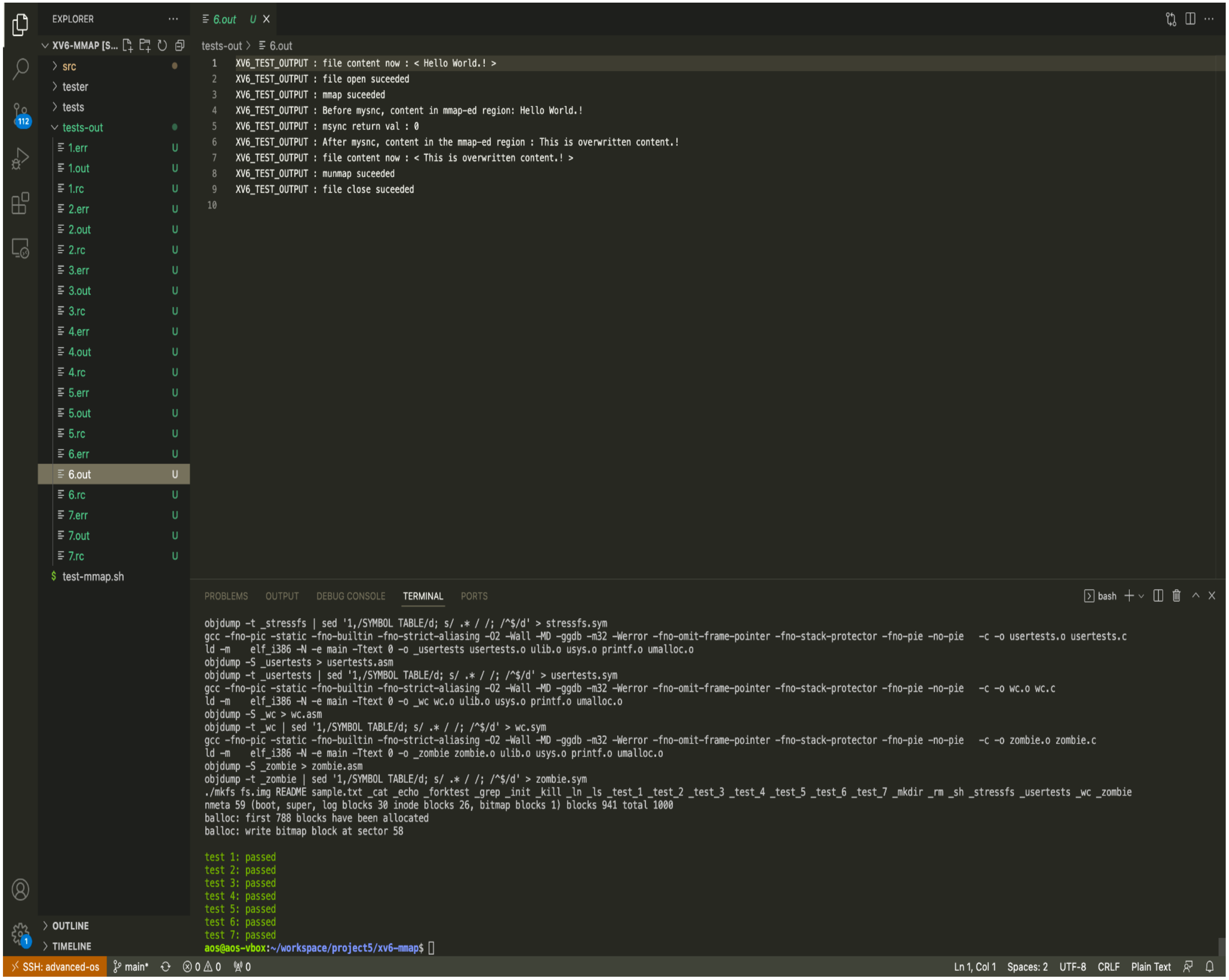
To implement lazy page allocation I first modified mmap to not allocate the memory for the requested region. The only thing that mmap does now is find a suitable virtual memory region for the request, allocate kernel memory for the mmap region structure, and set the metadata for that specific region (store the flags, start address, etc). Now, when the user tries to access that memory region, it will trigger a page fault. The next thing I did was handle this page fault. In trap.c I added a page fault handler in which I first check what triggered the page fault. If the page fault was caused by a write protection violation, then I killed the process and returned. If it was not caused by a write protection violation, then I traversed through the mmap regions linked list to find the region the faulting address belongs to. If it did not belong to any of the mmap regions, then I killed the process. If a region was found, then I allocated a page of physical memory and mapped it to the page that the faulting address falls within.

To implement file backed mappings, I first modified mmap to check if the request is for a file mapping. If it was, then I error checked the fd to make sure it was valid and then I duplicated it by calling filedup to increase the ref count. I stored the duplicated fd in the mmap region struct. The last thing I did was call fileseek to set the file offset to the one that was passed in. Since it was now using lazy page allocation, the file reading was actually now done in the page fault handler. To do this, I added logic to the page fault handler so that for file backed mappings it copied a PGSIZE chunk of the file into the newly allocated page (from part 1). Once it copied the contents of the file into the allocated memory, then it cleared the dirty bit belonging to that page table entry. I added the #define for the dirty bit in the mmu.h file (PTE_D 0x06). One last thing that I did to fully implement this part was to make sure to call fileclose in munmap to decrease the ref count for that file. Finally, to implement msync, I first searched in the mmap regions linked list for the region the address that was passed in belonged to. If it did not belong to one of the regions, then I returned -1. Once I found the region, I fileseeked to the offset that was passed in in mmap for the specific region. Then, starting with the starting address for the region, page by page, I searched for the pte for each page by calling walkpgdir. If no pte was found, then I continued onto the next page. If one was found, then I checked its dirty bit. If it was set, then I wrote the contents of that memory region to the file. Each time I checked a page, I made sure to fileseek to the next PGSIZE region of the file to keep it in sync with the page I was checking.

To implement fileseek, I first acquired the inode lock, then I changed the offset in the file struct, and then I released the inode lock.

NEXT PAGE CONTAINS A SCREENSHOT OF MY PASSING TESTS IN CASE I DIDN'T CREATE THE DIFF CORRECTLY



```
XV6_TEST_OUTPUT : file content now : < Hello World.! >
XV6_TEST_OUTPUT : file open suceeded
XV6_TEST_OUTPUT : mmap suceeded
XV6_TEST_OUTPUT : Before mysnc, content in mmap-ed region: Hello World.!
XV6_TEST_OUTPUT : msync return val : 0
XV6_TEST_OUTPUT : After mysnc, content in the mmap-ed region : This is overwritten content.!
XV6_TEST_OUTPUT : file content now : < This is overwritten content.! >
XV6_TEST_OUTPUT : munmap suceeded
XV6_TEST_OUTPUT : file close suceeded
```

```
objdump -t _stressfs | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > stressfs.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o usertests.o usertests.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _usertests usertests.o ulib.o usys.o printf.o umalloc.o
objdump -S _usertests > usertests.asm
objdump -t _usertests | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > usertests.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o wc.o wc.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _wc wc.o ulib.o usys.o printf.o umalloc.o
objdump -S _wc > wc.asm
objdump -t _wc | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > wc.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o zombie.o zombie.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _zombie zombie.o ulib.o usys.o printf.o umalloc.o
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
./mkfs fs.img README sample.txt _cat _echo _forktest _grep _init _kill _ln _ls _test_1 _test_2 _test_3 _test_4 _test_5 _test_6 _test_7 _mkdir _rm _sh _stressfs _usertests _wc _zombie
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 788 blocks have been allocated
balloc: write bitmap block at sector 58

test 1: passed
test 2: passed
test 3: passed
test 4: passed
test 5: passed
test 6: passed
test 7: passed
aos@aos-vbox:~/workspace/project5/xv6-mmap$ 
```